Fundamenta Informaticae 95 (2009) 1–25
DOI 10.3233/FI-2009-184
IOS Press

A Multi-swarm Approach to Multi-objective Flexible Job-shop Scheduling Problems

Hongbo Liu[*†]
School of Information Science and Technology, Dalian Maritime University,
Dalian 116026, China.
lhb@dlut.edu.cn

Ajith Abraham
Machine Intelligence Research Labs - MIR Labs, http://www.mirlabs.org.
Auburn, Washington 98071 USA
ajith.abraham@ieee.org

Zuwen Wang
School of Electromechanics and Materials Engineering, Dalian Maritime University,
Dalian 116026, China.
wangzw@dlmu.edu.cn

Abstract. Swarm Intelligence (SI) is an innovative distributed intelligent paradigm whereby the collective behaviors of unsophisticated individuals interacting locally with their environment cause coherent functional global patterns to emerge. In this paper, we model the scheduling problem for the multi-objective Flexible Job-shop Scheduling Problems (FJSP) and attempt to formulate and solve the problem using a Multi Particle Swarm Optimization (MPSO) approach. MPSO consists of multi-swarms of particles, which searches for the operation order update and machine selection. All the swarms search the optima synergistically and maintain the balance between diversity of particles and search space. We theoretically prove that the multi-swarm synergetic optimization algorithm converges with a probability of 1 towards the global optima. The details of the implementation for the multi-objective FJSP and the corresponding computational experiments are reported. The results indicate that the proposed algorithm is an efficient approach for the multi-objective FJSP, especially for large scale problems.

[*]Address for correspondence: School of Information Science and Technology, Dalian Maritime University, Dalian 116026, China
[†]This work is supported partly by NSFC (60873054), DLMU (DLMU-ZL-200709)

Keywords: Swarm Intelligence, Emergence, Multi-objective Optimization, Flexible Job-shop Scheduling Problem, Particle Swarm Optimization, Multi-swarm, Probability, Convergence

1. Introduction

Flexible job-shop scheduling problem is an extension of the classical JSP, which allows an operation to be processed by any machine from a given set. It incorporates all the difficulties and complexities of its predecessor JSP and is more complex than JSP because of the additional need to determine the assignment of operations to the machines. The job shop is flexible, i.e. there are multiple job routes. The scheduling problem of a FJSP consists of a routing sub-problem, that is, assigning each operation to a machine out of a set of capable machines and the scheduling sub-problem, which consists of sequencing the assigned operations on all machines in order to obtain a feasible schedule minimizing a predefined objective function. It is quite difficult to achieve an optimal solution with traditional optimization approaches owing to the high computational complexity. But it is one of the most critical issues in the planning and managing of manufacturing processes. Many practical problems have an underlying job-shop structure, such as multiprocessor task scheduling, network routing, robotic cell scheduling, project scheduling, railway scheduling, air traffic control.

Particle Swarm Optimization (PSO) incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the intelligence is emerged [1, 2]. It has become the new focus of research recently [3, 4, 5, 6, 7, 8, 9]. As an algorithm, its main strength is its fast convergence, which compares favorably with many other global optimization algorithms. However, for some complex problems, it often demonstrates faster convergence speed in the first phase of the search, and then slows down or even stops as the number of generations is increased. Once the algorithm slows down, it is difficult to achieve better fitness values. This state is called as stagnation or premature convergence. It is found that the swarm tends to collapse too fast when the process converges. In this paper, a multi-swarm PSO is investigated for solving the multi-objective FJSP. We introduce multi-swarms of particles to map different orders in the multi-objective FJSP, in which particles search for operation order update while others search for machine selection. All swarms search the optima synergistically and maintain the balance between diversity of particles and search space. The details of implementation for the multi-objective FJSP and the corresponding computational experiments are reported in this paper.

The rest of the paper is organized as follows. Related works about FJSP is reviewed in Section 2. We analyze the main factors of the multi-objective FJSP and formulate them in Section 3. In Section 4, particle swarm models are introduced and the MPSO model is described in detail. In Section 5, we theoretically prove the properties related to the encoding representation and the convergence of the proposed algorithm. Experiment settings, results and discussions are given in Section 6. Finally Conclusions are given in Section 7.

2. Related works

Flexible Job-shop Scheduling Problem has been drawing researchers' attention worldwide, not only because of its practical and theoretical importance, but also because of its complexity. The FJSP is a

NP-hard optimization problem [10, 11, 12]. Different approaches have been proposed to solve this problem. Bruker and Schlie [13] illustrated a polynomial algorithm for solving flexible job shop scheduling problems with two jobs. When the number of machines and the maximum number of operations per job are fixed, Jansen *et al.* [14] provided a linear time approximation scheme. Mastrolilli and Gambardella [15] proposed some neighborhood functions for FJSP. Because of the intractable nature of the problem and its importance in both fields of practical application and combinatorial optimization, it is desirable to explore other avenues for developing heuristic and metaheuristic algorithms. Pezzellaa *et al.* [16] classified them into two main categories: hierarchical approach and integrated approach. The hierarchical approach attempts to solve the problem by decomposing it into a sequence of subproblems, with reduced difficulty. Job routing and sequencing are usual studied separately [17, 18, 19, 20, 21, 22]. Most of them solve the assignment problem using some dispatching rules, and then solve it using different tabu search heuristics. Integrated approach is much more difficult to solve, but in general achieves better results. Hurink *et al.* [23] developed tabu search algorithms to solve the problem. Dauzére-Pérés and Paulli [24] extended the classical disjunctive graph model for job shop scheduling to take into account the fact that operations have to be assigned to machines in the FJSP. Based on the extended disjunctive graph, a new neighborhood structure is defined and a tabu search procedure is provided to solve the problem. Brandimarte [20], Mastrolilli and Gambardella [25], Saidi-Mehrabad and Fattahi [26] presented tabu search algorithms that solve the flexible job shop scheduling problem to minimize the makespan time.

More researchers attempt to solve the FJSP using genetic algorithms [16, 27, 28, 29]. Zhang and Gen [30] proposed a multistage operation-based GA to deal with the problem from a point view of dynamic programming. Chen *et al.* [31] split the chromosome representation into two parts, the first defining the routing policy, and the second the sequence of operations on each machine. Jia *et al.* [32] proposed a modified GA able to solve distributed scheduling problems and can be adapted for FJSP. Ho and Tay [33] proposed an efficient methodology called GENACE based on a cultural evolutionary architecture for solving FJSP with recirculation. Ho *et al.* [34] proposed an architecture for learning and evolving of Flexible Job-Shop schedules to improve the computational time and quality of schedules. Ong *et al.* [35] applied the clone selection principle of the human immune system to solve FJSP with re-circulation.

In many real-world FJSP, it is often necessary to optimize several criteria [36]. Minimization of makespan, lateness, tardiness, flow time, machine idle time, and such others are unusual the important criteria in the problems. Kacem *et al.* [37, 38] study on modeling genetic algorithms for FJSP. Tanev *et al.* [39] investigate an evolutionary algorithm-based approach for scheduling of customers' orders in factories of plastic injection machines (FPIM) as a case of real-world flexible job shop scheduling problem. They attempt to develop an efficient scheduling routine for planning the assignment of the submitted customers' orders to FPIM machines. The results obtained for evolving a schedule of 400 customers' orders on experimental model of FPIM indicate that the business delays in order of half-an-hour can be achieved.

Recently, swarm intelligence and multiagent techniques have attracted the attention of several researchers from different application domains. Liouane *et al.* [40] proposed a hybrid algorithm based on ant systems and local search optimization for FJSP. Blum and Samples [41] illustrated a neighborhood structure for the problem by extending the well-known neighborhood structure derived by Nowicki and Smutnicki [42] for the job shop scheduling problem. Then, the authors developed an ant colony optimization approach, which uses a strong non-delay guidance for constructing solutions and which employs black-box local search procedures to improve the constructed solutions. Wu and Weng [43] proposed a multi-agent scheduling method with job earliness and tardiness objectives in a flexible job-shop

environment. The computational experiments show that the proposed multi-agent scheduling method is quite fast. By hybridizing particle swarm optimization and simulated annealing, Xia and Wu [44] developed a hybrid approach for the multi-objective flexible job-shop scheduling problem. Sha and Hsu [45] applied Giffler and Thompson's heuristic [46] to decode a particle position into a schedule. The computational results show that their approaches can obtain better solutions. They also point out the future works: (1) modify particle position representation for better suitability to the problem; (2) design other particle movement methods and particle velocity for the modified particle position representation. As Baykasoğlu *et al* [47] discussed, the most important issue in employing meta-heuristics for combinatorial optimization problems is to develop an effective "problem mapping" and "solution generation" mechanism. If these two mechanisms are devised successfully, then it is possible to find good solutions to a given optimization problem in an acceptable time.

3. Problem and Formulation

We focus on flexible job-shop scheduling problems composed of the following elements:

- Jobs. $J = \{J_1, \cdots, J_n\}$ is a set of n jobs to be scheduled. Each job J_i consists of a predetermined sequence of operations. $O_{i,j}$ is the operation j of J_i. All jobs are released at time 0.
- Machines. $M = \{M_1, \cdots, M_m\}$ is a set of m machines. Each machine can process only one operation at a time. And each operation can be processed without interruption during its performance on one of the set of machines. All machines are available at time 0.
- Flexibility. The multi-objective FJSP usually is classified into two types as follows:
  - Total FJSP (T-FJSP): each operation can be processed on any machine of M.
  - Partial FJSP (P-FJSP): each operation can be processed on one machine of subset of M.
- Constraints. The constraints are rules that limit the possible assignments of the operations. They can be divided mainly into following situations:
  - Each operation can be processed by only one machine at a time (disjunctive constraint).
  - Each operation, which has started, runs to completion (non-preemption condition).
  - Each machine performs operations one after another (capacity constraint).
  - Although there are no precedence constraints among operations of different jobs, the predetermined sequence of operation for each job forces each operation to be scheduled after all predecessor operations (precedence/conjunctive constraint).
  - the machine constraints emphasize the operations can be processed only by the machine from the given set (resource constraint).
- Objective(s). Most of the research reported in the literature is focused on the single objective case of the problem, in which the objective is to find a schedule that has minimum time required to complete all operations (minimum makespan). Some other objectives, such as flow time or tardiness are also important like the makespan. Currently it has been paid more attentions to investigate the problem from a multiobjective perspective. It is desirable to generate many near-optimal schedules considering multiple objectives.

To formulate the objective, define $C_{i,j,k}$ ($i = 1, 2, \cdots, n$; $j = 1, 2, \cdots, a$; $k = 1, 2, \cdots, m$) as the completion time that the machine M_k finishes the j-th operation $O_{i,j}$ of job i; $\sum C_k$ represents the time that the machine M_k completes the processing of all the assigning jobs. Define $C_{sum} = \sum_{k=1}^{m}(\sum C_k)$ as the flowtime, and $C_{max} = max\{\sum C_k\}$ as the makespan. The problem is thus to both determine an assignment and a sequence of the operations on all machines that minimize the criteria:

- The sum of the completion times (flowtime): C_{sum}.
- the maximum completion time (makespan): C_{max}.

Let $A(t)$ be the set of operations being processed at time t, and let $r_{i,j,k} = 1$ if operation j of job i is assigned on machine k to be processed and $r_{i,j,k} = 0$ otherwise. Let $d_{i,j}$ denote the duration (processing time) of operation j of job i. The conceptual model of the multi-objective FJSP can be formulated the following way:

$$\textbf{Minimize} \quad f(C_{max}, C_{sum}) \tag{1}$$

subject to :

$$C_{i,j,k} \leq C_{i,j+1,k} - d_{i,j+1}. \quad j = 1, \cdots, a-1. \tag{2}$$

$$\sum_{j \in A(t)} r_{i,j,k} \leq 1, \quad k \in M; t \geq 0. \tag{3}$$

$$C_{i,j,k} \geq 0, \quad i = 1, \cdots, n. \tag{4}$$

Minimizing C_{sum} asks the average job finishes quickly, at the expense of the largest job taking a long time, whereas minimizing C_{max}, asks that no job takes too long, at the expense of most jobs taking a long time. Minimization of C_{max} would result in maximization of C_{sum}. The weighted aggregation is the most common approach to the problems. According to this approach, the objectives, $f_1 = \min\{C_{sum}\}$ and $f_2 = \min\{C_{max}\}$, are summed to a weighted combination:

$$f = \min(w_1 f_1 + w_2 \lambda f_2) \tag{5}$$

where λ is the scaling factor, which is the average number of machines per operation; w_1 and w_2 are non-negative weights, and $w_1 + w_2 = 1$. These weights can be either fixed or adapt dynamically during the optimization [48]. The fixed weighted aggregation $(1/2)$ is used in the paper. Alternatively, the weights can be changed gradually according to the Eqs. (6) and (7). The variation for different values of w_1 and w_2 ($R = 200$) are illustrated in Fig. 1.

$$w_1(t) = |sin(2\pi t/R)| \tag{6}$$

$$w_2(t) = 1 - w_1(t) \tag{7}$$

Definition 3.1. A flexible job-shop scheduling problem can be defined as $\prod = (J, O, M, f)$. The key components are jobs, operations and machines. For the sake of simplify, the scheduling problem also be represented in triple $P = (J, O, M)$.

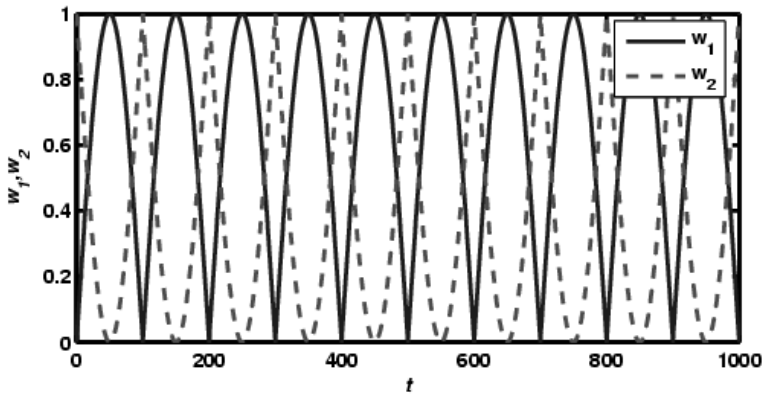

Figure 1. Dynamic weight variation.

The complexity of FJSP increases with the number of constraints imposed and the size of search space employed [49]. Except for some highly restricted special cases, very simple special cases of FJSP are already strongly NP-hard. For the FJSP, the size of search space is $(n!)^m$, and for this reason, it is computationally infeasible to try every possible solution. This is because the required computation time increases exponentially with the problem size. In practice, many real-world FJSPs have a larger number of jobs and machines as well as additional constraints and flexibilities, which further increase its complexity. For the same number of machines and jobs, the P-FJSP is more difficult to solve than the T-FJSP. Therefore, the P-FJSP is transformed to the T-FJSP by adding 'infinite processing times' to the unused machines and to solve the latter instead in [37]. However, although the P-FJSP is a generalization of the T-FJSP, Ho *et al*. illustrated the distinguish between the problem types of T-FJSP and P-FJSP [34].

4. PSO algorithms for FJSP

For applying the particle swarm algorithm successfully for any problem, one of the key issues is how to map the problem solution to the particle space, which affects its feasibility and performance [50]. We introduce a novel multi-swarm approach to explore the better solutions for the FJSP. In this section, we firstly review PSO briefly and discuss how to solve the FJSP using multi-swarm algorithms.

4.1. Review of standard PSO

The standard PSO model consists of a swarm of particles moving in a d-dimensional search space where the fitness f can be calculated as a certain quality measure. Each particle has a position represented by a position-vector $\vec{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{p}_i$, and its j-th dimensional value is $p_{i,j}$. The best position from the swarm thus far is then stored in a vector $\vec{p}^*$, and its j-th dimensional value is p_j^*. During the iteration time t, the update of the velocity from the previous velocity is determined by Eq. (8). Subsequently, the new position is determined by the sum of the previous position and the new

velocity by Eq. (9).

$$\begin{aligned} v_{i,j}(t) = wv_{i,j}(t-1) + c_1 r_1(p_{i,j}(t-1) - x_{i,j}(t-1)) \\ + c_2 r_2(p_j^*(t-1) - x_{i,j}(t-1)) \end{aligned} \tag{8}$$

$$x_{i,j}(t) = x_{i,j}(t-1) + v_{i,j}(t) \tag{9}$$

where r_1 and r_2 are the random numbers, uniformly distributed within the interval [0,1] for the j-th dimension of i-th particle. c_1 is a positive constant termed as the coefficient of the self-recognition component; c_2 is a positive constant termed as the coefficient of the social component. The variable w is the inertia factor, for which value is typically setup to vary linearly from 1 to 0 during the iterated processing. From Eq. (8), a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. In the particle swarm model, the particle searches the solutions in the problem space within a range $[-s, s]$ (If the range is not symmetrical, it can be translated to the corresponding symmetrical range.) In order to guide the particles effectively in the search space, the maximum moving distance during one iteration is clamped in between the maximum velocity $[-v_{max}, v_{max}]$ given in Eq. (10), and similarly for its moving range given in Eq. (11):

$$v_{i,j} = sign(v_{i,j}) min(|v_{i,j}|, v_{max}) \tag{10}$$

$$x_{i,j} = sign(x_{i,j}) min(|x_{i,j}|, x_{max}) \tag{11}$$

The value of v_{max} is $\rho \times s$, with $0.1 \leq \rho \leq 1.0$ and is usually chosen to be s, i.e. $\rho = 1$. The pseudo-code for particle-search is illustrated in Algorithm 1. The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region which is defined by each individual's previous best success and the success of some other particle. Bergh and Engelbrecht [51] overviewed the theoretical studies, and extend these studies to investigate particle trajectories for general swarms to include the influence of the inertia term. They also provided a formal proof that each particle converges to a stable point. It has been shown that the trajectories of the particles oscillate as different sinusoidal waves and converge quickly. Liu and Abraham [52] analyze the chaos and effects of the change in the velocities of particles. Eberhart and Kennedy called the two basic methods as "$g$best model" and "$l$best model" [1]. In the $l$best model, particles have information only of their own and their nearest array neighbors' best, rather than that of the whole swarm. Namely, in Eq. (8), $g$best is replaced by $l$best in the model. The $l$best model allows each individual to be influenced by some smaller number of adjacent members of the population array. The particles selected to be in one subset of the swarm have no direct relationship to the other particles in the other neighborhood. Typically $l$best neighborhoods comprise exactly two neighbors. When the number of neighbors increases to all but itself in the $l$best model, the case is equivalent to the $g$best model. Unfortunately there is a large computational cost to explore the neighborhood relation in each iteration. In the $g$best model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Some previous studies has been shown that the trajectories of the particles oscillate in different sinusoidal waves and converge quickly in the "$g$best model" algorithm [53, 54]. During the iteration, the particle is attracted towards the location of the best fitness achieved so far by the particle itself and by the location of the best fitness achieved so far across the whole swarm [55, 56].

Algorithm 1 Particle Swarm Algorithm

```
01. Begin
02.   Parameter settings and initialize swarm
03.   Evaluation
04.   Locate leader
05.   t = 1
06.   While (the end criterion is not met) do
07.       For each particle
08.           Update velocity
09.           Update position
10.           Evaluation
11.           Update pbest
12.       EndFor
13.       Update leader
14.   t++
15.   End While
16. End
```

4.2. Encoding Representations and Decoding Alignment

Encoding representation can be extremely important when trying to find solutions to a problem in a heuristic or metaheuristic algorithm. The data structures, such as the particle position, plus the algorithm combine to make efficient programs. Better efficiency of search can be achieved by modifying the encoding representation and its related operators so as to generate feasible solutions and avoiding the use of a repair mechanism. A bad encoding representation can increase the size of the search space or slow down the algorithm if too many repair operators are needed to ensure the representation is valid.

Cheng *et al.* [27], Kleeman and Lamont [36] introduced the taxonomy of how EAs represent job-shop problems. These representations can be classified as either directly encoded approaches or indirectly coded approaches. With a direct approach, a schedule is encoded into the chromosome. The EA then operates on these schedules in an effort to find the best schedule. For direct approaches, there are five different ways the EA can be encoded:

- Operation-based
- Job-based
- Job pair relation-based
- Completion time-based
- Random keys

Indirect approaches are chromosome representations that do not directly encode the schedule into the chromosome. There are four indirect approaches:

- Preference list-based
- Priority rule-based

Table 1. An example of the P-FJSP.

		M_1	M_2	M_3
J_1	$O_{1,1}$	4	5	XXX
	$O_{1,2}$	9	2	2
	$O_{1,3}$	XXX	6	3
J_2	$O_{2,1}$	6	5	XXX
	$O_{2,2}$	3	3	5

- Disjunctive graph-based
- Machine-based

For encoding representations, we have to consider time and space computational complexity, the need to maintain solution feasibility. To solve the FJSP, there are the three important factors:

- Flexible
- Length
- Availability

There are an fixed sequence for operations in each job (precedence constraints). But there are no precedence constraints among operations of different jobs. If the job and the operations are ordered beforehand, there is not enough flexible between jobs. Gen *et al.* [57] proposed a perfect method: they name all operations for a job with the same symbol (for example, the corresponding job index) and then interpret them according to the order of occurrences in the sequence of a given chromosome. All permutations of the chromosome yield a valid schedule. The chromosome length is $\sum_{i=1}^{n} a_i$, where n is the number of jobs and a_i is the number of operations in job i. It is possible for this kind of representation to cause some invalid candidate solutions after crossover and mutation operators. For example, one initial chromosome, $\{0, 1, 0, 1, 0\}$, is valid for the problem in Table 1 (0s denote the operations of J_1, and 1s denote the operations of J_2). A mutation operator bring out a invalid one, $\{0, 1, 1, 1, 0\}$ or $\{0, 1, 0, 1, 1\}$. In the new chromosome, the number of the candidate symbol, 0, is less than the number of operations in J_1. And the number of the candidate symbol, 1, is more than the number of operations in J_2. It would also be confronted with another difficulty: how to check effectively which machine can process the assigning operations when it deals with P-FJSP. For the same number of machines and jobs, Kacem *et al.* [37] transformed the P-FJSP to the T-FJSP by adding 'infinite processing times' to the unused machines and to solve it. Some individuals would be evaluated to 'infinite'. This increases the overall time complexity due to the presence of redundant assignments. Ho *et al.* [34] proposed a new chromosomal representation, which has two components: operation order and machine selection. Operation order component is similar to Gen's method. Each individual is obtained from this schedule by replacing each operation by the corresponding job index. By reading the data from left to right and increasing operation index of each job, a feasible schedule is always obtained. The machine selection component consists of a chromosome of size $\sum_{i=1}^{n} a_i$. Each allele of the chromosome is a sub-chromosome, which lists the preference which machine would process the operation. For the problem in Table 1, one possible encoding is shown in Fig. 2. This method inherits the advantages of both the operation-based chromosome representation and

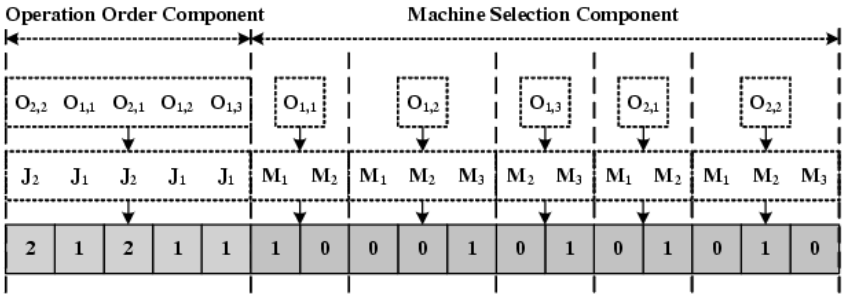

Figure 2. Operation-Order-Machine-Selection representation.

the preference list-based representation. The chromosome length is $\sum_{i=1}^{n} a_i + \sum_{i=1}^{n} a_i * b_j$, where n is the number of jobs, a_i is the number of operations in job i, b_j is the number of machines which operation $O_{i,j}$ can be assigned on. This chromosome representation has to consider the availability of machines that process operations so that the decoding processing reduces the search space size. But it is possible that one operation is assigned on more than one machine after crossover and mutation operators. Therefore, a repair mechanism to maintain feasibility is required. In addition, this representation is complex and redundant for the T-FJSP, since the machine selection component seems too long.

Due to the continuous characters of the positions of particles in standard PSO model, its encoding scheme cannot be directly adopted for the FJSP. So one of the most important problems in applying PSO to FJSP is to find how to map the problem solution to the particle. In our PSO algorithms, the position representation of the particles also has two components: operation order and machine selection. But it is with a variable length strategy.

The first part, operation order component provides the order of operations. For convenience, we will decompose all the jobs to atomic operations, and all operations for a job is signed with the corresponding job index. Then we map all the operations and jobs to the particles' positions. The positions are ranked to the corresponding job index by incorporating Ranked Order Value (ROV) rule [58] based on the Smallest Position Value (SPV) encoding rule [59]. In the ROV rule, the first SPV(s) of a particle is handled and assigned a smallest rank value 1(s). Then, the SPV(s) will be handled and assigned a rank value 2(s). With the same way, all the position values will be dealt with to convert the position information of a particle to a job permutation. There are two jobs, three operations in job 1, and two operations in job 2 as shown in Table 1 to illustrate the rank rule. In the instance ($n = 2$, $p_1 = 3$, $p_2 = 2$), position information is $X_i = [2.9, 0.6, 3.7, 1.8, 1.2]$. Because $x_{i,2}$, $x_{i,5}$, $x_{i,4}$ is the first three SPV of the particle, they are handled firstly and assigned rank value 1 as the job index of job 1, then the remain SPV, $x_{i,1}$ and $x_{i,3}$ are assigned rank value 2 as the job index of job 2. Thus, the operation order is obtained, i.e., $\{O_{2,1}, O_{1,1}, O_{2,2}, O_{1,2}, O_{1,3}\}$ as shown in Fig. 3.

The second part, machine selection component is variable length according to the problems. If the Average number of Machines per Operation (AMO) is larger than the half of number of machines, we encode each dimension with a random number in the interval $[1, m+1)$. Each dimension of the particle's position maps one operation, and the value of the position indicates the machine number to which this task/operation is assigned during the course of particle swarm algorithm. So the value of a particle's position should be integer. But after updating the velocity and position of the particles, the particle's position may appear real values such as 1.4, etc. It is meaningless for the assignment. Therefore, in

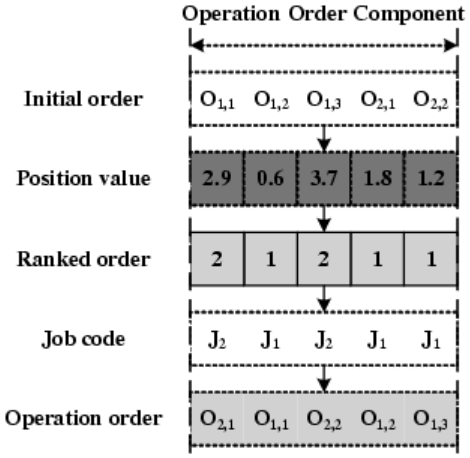

Figure 3. Representation of position values and the corresponding operation order.

the algorithm we usually round off the real optimum value to its nearest integer number. In this part, the sequence of the operations will be changed during the iteration according to the first part, operation order component. The feasible different sequence schedule of the operations between different jobs comes from the operation order. If AMO is less than or equal to the half of number of machines, we extend it to feasible machine representation. According to the operation order, each operation has a sub-sequence for machine selection, which lists the preference which machine would process the operation. The first SPV of the sub-sequence is assigned a rank value 1, other(s) are 0. The corresponding machine is selected if its value is 1. Since the AMO of the P-FJSP in Table 1 is 2.4, larger than 1.5, its machine selection component can be encoded as shown in Fig. 4. Only for demonstration, we also use the same sample to illustrate our encoding representation as shown in Fig. 5. The variable length representation allows the algorithm to maintain a balance between the flexibility of FJSP and search space, and to converge on the better results effectively.

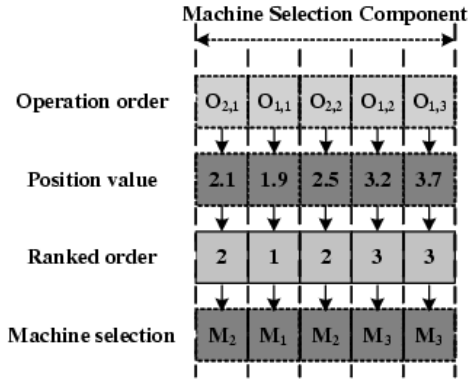

Figure 4. Representation of position values and the corresponding single-machine selection.

The standard PSO algorithm uses a swarm of particles. In the search process, particles are supposed to follow the best particle from the swarm. As above remember, its performance deteriorates as the di-

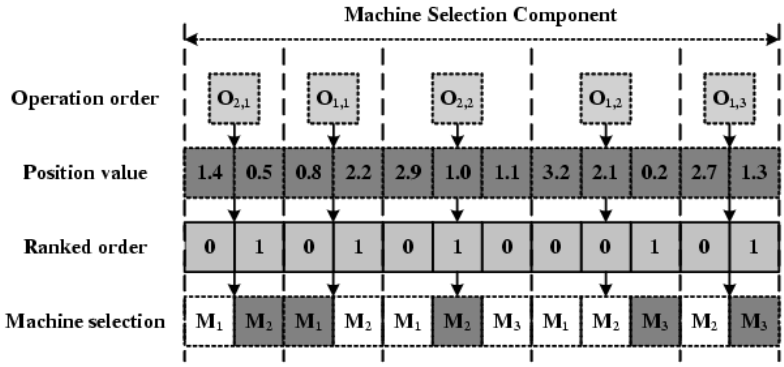

Figure 5. Representation of position values and the corresponding multi-machine selection.

mensionality of the search space increases, especially for the multi-objective FJSP involving large scale. PSO often demonstrates faster convergence speed in the first phase of the search, and then slows down or even stops as the number of generations is increased. Once the algorithm slows down, it is difficult to achieve better scheduling solutions. Bergh and Engelbrecht [60] investigated effects of swarm size on cooperative particle swarm optimizations. They proposed a multi-swarm cooperative particle swarm optimizer, which takes the n-dimensional solution vector and breaks it into n one-dimensional components. Each component is then optimized by a separate PSO. The objective function is evaluated using a vector formed by concatenating the components from the n swarms to again form an n-dimensional vector. The algorithm forms solution vectors by combining different vectors from different swarms, effectively creating more diversity out of fewer particles. There is more better balance between the dimension and the number of iterations. Grosan *et al.* [4] divided the swarm of PSO into multiple independent sub-swarms so as to obtain multiple different points for the geometrical place problems. By considering different sub-swarms, the number of solutions which can be obtained at the end of the search process might be at most equal to the number of sub-swarms. The algorithm is successful to solve the geometrical place problems. Since our encoding representation consists of two components, we split the swarm of particles into two independent sub-swarms for the FJSP. The operation order component is mapped to the first sub-swarm, which takes the b-dimensional solution vector ($b = \sum_{i=1}^{n} a_i$). The second sub-swarm deals with the machine selection component. The two sub-swarms search the optima cooperatively and maintain the balance between diversity of particles and search space.

In our encoding representations, we can consider particle's position encoding as the binary representation of an integer. And the step size is equal to 1, i.e., the dimension of the search space is then 1. In practice, when the binary string is too long for a large scale problem, it has too high dimension for us to use it as an integer. It is time-consuming for each iteration. So we split it into a small number (say H) of shorter binary strings, each one is seen as an integer. Then the dimension of the problem is not anymore 1, but H. The swarm algorithm with two strategies is so called as Bi-metrics Binary PSO. Fig. 6 illustrates the direct encoding representation. And Fig. 7 illustrates the compositive encoding representation.

Since the particle's position indicates the potential schedule, the position can be "decoded" to the scheduling solution. It is to be noted that the solution will be unfeasible if it violates the sequence constraint (2). The operations must be started only after the completion of the previous latest operation in the sequence operation. The best situation is the starting point of the operation in alignment with the

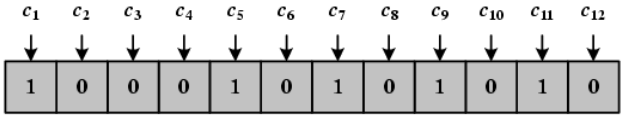

Figure 6. Bi-metrics Binary Representation - Direct encoding.

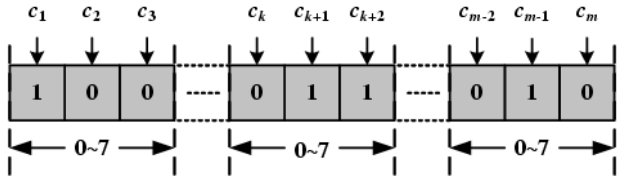

Figure 7. Bi-metrics Binary Representation - Compositive encoding.

ending point of its previous latest operation. After all the operations have been processed, we get the feasible scheduling solution and then calculate the cost of the solution.

4.3. Multi-Swarm PSO

To employ a multi-swarm the solution vector is split amongst the different populations according to some rule; the simplest of the schemes does not allow any overlap between the spaces covered by different populations. To find a solution to the original problem, representatives from all the populations are combined to form the potential solution vector, which, in turn, is passed on to the error function. This adds a new dimension to the survival game: cooperation between different populations [51, 63, 64].

The different individual is separated into different group to map the operation order and machine selection respectively, which is favorable and reasonable. To match the two component characteristics, we introduce a multi-swarm search algorithm for them. In the algorithm, all particles are clustered spontaneously into two different groups of the whole swarm. One is mapped to the operation order, and another to the machine selection. Each group consists of multiple sub-swarms. In the same group, every particle can connect more than one sub-swarm, and a crossover neighborhood topology is constructed between different sub-swarms. The particles in the same sub-swarm would carry some similar functions as possible and search their optimal. Each sub-swarm would approach to its appropriate position (solution), which would be helpful for the whole swarm to keep in a good balance state. Fig. 8 illustrates a multi-swarm topology. In the swarm system, a swarm with 30 particles is organized into 10 sub-swarms, which one consists of 5 particles. Particles 3 and 13 have the maximum membership level, 3. During the iterated process, the particle updates its velocity following by the location of the best fitness achieved so far by the particle itself and by the location of the best fitness achieved so far across all its neighbors in all sub-swarms it belongs to. We consider the multi-swarm algorithm more about operation order update and others for machine selection in the multi-objective Flexible Job-shop Scheduling Problems. The multi-swarm algorithm for FJSP is illustrated as follows:

Step 1. Given s swarms, n particles in each swarm, encode the first $s/2$ swarms according to the operation order, and encode other swarms according to the machine selection. Initialize the positions and the velocities for all the particles randomly. Initialize other parameters.

Step 2. For the operation order swarms and machine selections the multiple sub-swarms n are organized respectively into a crossover neighborhood topology. A particle can join more than one sub-swarm. Each particle has the maximum membership level l, and each sub-swarm accommodates default number of particles m.

Step 3. Decode the positions and evaluate the fitness for each particles.

Step 4. Find the best particle in the swarm, and find the best one in each sub-swarms. If the "global best" of the swarm is improved, $noimprove = 0$, otherwise, $noimprove = 1$. Update velocity and position for each particle at the iteration t.

4.01 For m = 1 to $subs$
4.02 $\vec{p}^{*} = argmin_{i=1}^{subs_m}(f(\vec{p}^{*}(t-1)), f(\vec{x}_1(t)),$
4.02 $f(\vec{x}_2(t)), \cdots, f(\vec{x}_i(t)), \cdots, f(\vec{x}_{subs_m}(t)))$;
4.03 For ss = 1 to $subs_m$
4.04 $\vec{p}_i(t) = argmin(f(\vec{p}_i(t-1)), f(\vec{x}_i(t))$;
4.05 For d = 1 to D
4.06 Update the d-th dimension value of $\vec{x}_i$ and $\vec{v}_i$
4.06 according to Eqs. (8), (10), (9), and (11);
4.07 Next d
4.08 Next ss
4.09 Next m

Step 5. If $noimprove = 1$, goto Step 2, the topology is re-organized. If the end criterion is not met, goto Step 3. Otherwise, output the best solution, the fitness.

5. Algorithm analysis

For analyzing the convergence of the multi-swarm algorithm, we first introduce the definitions and lemmas [65, 66, 67], and then theoretically prove that the algorithm converges with a probability 1 or strongly towards the global optimal.

Xu, *et al* [68] analyzed the search capability of an algebraic crossover through classifying the individual space of genetic algorithms, which is helpful to comprehend the search of genetic algorithms such that premature convergence and deceptive problems [69] could be avoided. In this subsection, we also attempt to theoretically analyze the performance of the multi-swarm algorithm with crossover neighborhood topology. For the sake of convenience, let crossover operator $|_c$ denote the wheeling-round-the-best-particles process.

Consider the problem (P) as

$$(P) = min\{f(\vec{x}) : \vec{x} \in D\} \tag{12}$$

where $\vec{x} = (x_1, x_2, \cdots, x_n)^T$, $f(\vec{x}) : D \to R$ is the objective function and D is a compact Hausdorff space. Applying our algorithm the problem (P), it can be transformed to P' as

$$(P') = \begin{cases} minf(\vec{x}) \\ \vec{x} \in \Omega = [-s, s]^n \end{cases} \tag{13}$$

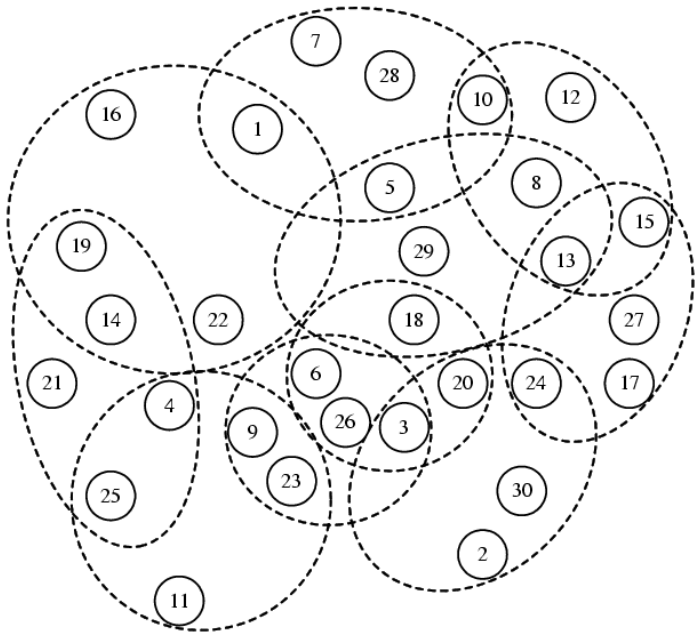

Figure 8. A multi-swarm topology.

where Ω is the set of feasible solutions of the problem. A swarm is a set, which consists of some feasible solutions of the problem. Assume S as the encoding space of D. A neighborhood function is a mapping $\mathcal{N} : \Omega \to 2^{\Omega}$, which defines for each solution $S \in \Omega$ a subset $\mathcal{N}(S)$ of Ω, called a neighborhood. Each solution in $\mathcal{N}(S)$ is a neighbor of S. A local search algorithm starts off with an initial solution and then continually tries to find better solutions by searching neighborhoods [15]. Most generally said, in swarm algorithms the encoding types S of particles in the search space D are often represented as strings of a fixed-length L over an alphabet. Without loss of generality, S can be described as

$$S = \underbrace{z_m \times \cdots \times z_m}_{L} \tag{14}$$

where z_m is a finite field about integer number $\mod m$. Most often, it is the binary alphabet, *i.e.* $m = 2$.

Proposition 5.1. If k alleles are '0's in the nontrivial ideal Ω, *i.e.* $L - k$ alleles are uncertain, then θ_Ω partitions Ω into 2^k disjoint subsets as equivalence classes corresponding to Holland's schema theorem [70, 71], *i.e.*, each equivalence class consists of some '1's which k alleles in Ω with '0' are replaced by '1's. Let $A \in S/\theta_\Omega$, then there is an minimal element m of A under partial order $(S, \vee, \wedge, \neg)$, such that $A = \{m \vee x \mid x \in \Omega\}$.

Theorem 5.1. Let A, B, C are three equivalence classes on θ_Ω, where θ_Ω is the congruence relation about Ω. $\exists\, x \in A$, $y \in B$, and $x \mid_c y \in C$, then $C = \{x \mid_c y \mid x \in A, y \in B\}$.

Proof:
Firstly, we verify that for any $d_1, d_2 \in \Omega$, if $x \mid_c y \in C$, then $(x \vee d_1) \mid_c (y \vee d_2) \in C$. In fact,

$$\begin{aligned}(x \vee d_1) \mid_c (y \vee d_2) =& (x \vee d_1)c \vee (y \vee d_2)\bar{c} \\ & (xc \vee y\bar{c}) \vee (d_1 c \vee d_2 \bar{c}) \\ & (x \mid_c y) \vee (d_1 c \vee d_2 \bar{c})\end{aligned} \tag{15}$$

Obviously, $(d_1 c \vee d_2 \bar{c}) \in \Omega$, so $(x \vee d_1) \mid_c (y \vee d_2) \equiv (x \mid_c y)(\mod \theta_\Omega)$, *i.e.* $(x \vee d_1) \mid_c (y \vee d_2) \in \Omega$.

Secondly, from Proposition 5.1, $\exists m, n, d_3, d_4 \in \Omega$ of A, B, such that $x = m \vee d_3$, $y = n \vee d_4$. As a result of analysis in Eq.(15), $x \mid_c y \equiv (m \mid_c n)(\mod \theta_\Omega)$, *i.e.*, $m \mid_c n \in C$.

Finally, we verify that $m \mid_c n$ is a minimal element of C and $(m \mid_c n) \vee d = (m \vee d) \mid_c (n \vee d)$. As a result of analysis in Eq.(15), if $d_1 = d_2 = d$, then $m \mid_c n \vee d = (m \vee d) \mid_c (n \vee d)$. Therefore $m \mid_c n$ is a minimal element of C.

To conclude, $C = \{(m \mid_c n) \vee d \mid d \in \Omega\} = \{x \mid_c y \mid x \in A, y \in B\}$. The theorem is proven. □

Proposition 5.2. Let A, B are two equivalence classes on θ_Ω, and there exist $x \in A$, $y \in B$, such that $x \mid_c y \in C$, then, $x \mid_c y$ makes ergodic search C while x and y make ergodic search A and B, respectively.

Definition 5.1. (Convergence in terms of probability)
Let ξ_n a sequence of random variables, and ξ a random variable, and all of them are defined on the same probability space. The sequence ξ_n converges with a probability of ξ if

$$\lim_{n\to\infty} P(|\xi_n - \xi| < \varepsilon) = 1 \tag{16}$$

for every $\varepsilon > 0$.

Definition 5.2. (Convergence with a probability of 1)
Let ξ_n a sequence of random variables, and ξ a random variable, and all of them are defined on the same probability space. The sequence ξ_n converges almost surely or almost everywhere or with probability of 1 or strongly towards ξ if

$$P\left(\lim_{n\to\infty} \xi_n = \xi\right) = 1; \tag{17}$$

or

$$P\left(\bigcap_{n=1}^{\infty} \bigcup_{k\geq n} [|\xi_n - \xi| \geq \varepsilon]\right) = 0 \tag{18}$$

for every $\varepsilon > 0$.

Theorem 5.2. Let $\vec{x}^*$ is the global optimal solution to the problem (P'), and $f^* = f(\vec{x}^*)$. Assume that the clubs-based multi-swarm algorithm provides position series $\vec{x}_i(t)$ $(i = 1, 2, \cdots, n)$ at time t by the iterated procedure. $\vec{p}^*$ is the best position among all the swarms explored so far, *i.e.*

$$\vec{p}^*(t) = arg \min_{1\leq i\leq n} (f(\vec{p}^*(t-1)), f(\vec{p}_i(t))) \tag{19}$$

Then,

$$P\left(\lim_{t\to\infty} f(\vec{p}^*(t)) = f^*\right) = 1 \tag{20}$$

Proof:
Let

$$D_0 = \{\vec{x} \in \Omega | f(\vec{x}) - f^* < \varepsilon\} \\ D_1 = \Omega \setminus D_0 \tag{21}$$

for every $\varepsilon > 0$.

While the different swarm searches their feasible solutions by themselves, assume Δp is the difference of the particle's position among different club swarms at the iteration time t. Therefore $-s \leq \Delta p \leq s$. $Rand(-1, 1)$ is a normal distributed random number within the interval [-1,1]. According to the update of the velocity and position by Eqs. (8)∼(9), Δp belongs to the normal distribution, *i.e.* $\Delta p \sim [-s, s]$. During the iterated procedure from the time t to $t + 1$, let q_{ij} denote that $\vec{x}(t) \in D_i$ and $\vec{x}(t + 1) \in D_j$. Accordingly the particles' positions in the swarm could be classified into four states: q_{00}, q_{01}, q_{10} and q_{01}. Obviously $q_{00} + q_{01} = 1$, $q_{10} + q_{11} = 1$. According to Borel-Cantelli Lemma and Particle State Transference [56], proving by the same methods, $q_{01} = 0$; $q_{00} = 1$; $q_{11} \leq c \in (0, 1)$ and $q_{10} \geq 1 - c \in (0, 1)$.

For $\forall \varepsilon > 0$, let $p_k = P\{|f(\vec{p}^*(k)) - f^*| \geq \varepsilon\}$, then

$$p_k = \begin{cases} 0 & \text{if } \exists T \in \{1, 2, \cdots, k\}, \vec{p}^*(T) \in D_0 \\ \bar{p}_k & \text{if } \vec{p}^*(t) \notin D_0, t = 1, 2, \cdots, k \end{cases} \tag{22}$$

According to Particle State Transference Lemma,

$$\bar{p}_k = P\{\vec{p}^*(t) \notin D_0, t = 1, 2, \cdots, k\} = q_{11}^k \leq c^k. \tag{23}$$

Hence,

$$\sum_{k=1}^{\infty} p_k \leq \sum_{k=1}^{\infty} c^k = \frac{c}{1 - c} < \infty. \tag{24}$$

According to Borel-Cantelli Lemma,

$$P\left(\bigcap_{t=1}^{\infty} \bigcup_{k \geq t} |f(\vec{p}^*(k)) - f^*| \geq \varepsilon\right) = 0 \tag{25}$$

As defined in Definition 5.2, the sequence $f(\vec{p}^*(t))$ converges almost surely or almost everywhere or with probability 1 or strongly towards f^*. The theorem is proven. □

6. Experiment Settings, Results and Discussions

The algorithm procedure described in Section 4 has been implemented on Intel Core® Duo™ CPU 1.73 GHz processor with 1G memory. To illustrate the effectiveness and performance of the proposed algorithm, three representative instances based on practical data have been selected. Three problem instances $(J8, O27, M8)$, $(J10, O30, M10)$ and $(J15, O56, M10)$ are taken from Kacem *et al*. [37, 38, 72]. In our experiments, the algorithms used for comparison were GA (Genetic Algorithm) [76, 77], SPSO (standard PSO) [1], and MPSO (Multi-swarm PSO). These algorithms share many similarities.

Table 2. Parameter settings for the algorithms.

Algorithm	Parameter name	Value
GA	Size of the population	$(even)(int)(10+2*sqrt(D))$
	Probability of crossover	0.8
	Probability of mutation	0.01
PSO(s)	Swarm size	$(even)(int)(10+2*sqrt(D))$
	Self coefficient c_1	$0.5+log(2)$
	Social coefficient c_2	$0.5+log(2)$
	Inertia weight w	0.91
	Clamping Coefficient ρ	0.5

GA is powerful stochastic global search and optimization methods, which are also inspired from the nature like the PSO. Genetic algorithms mimic an evolutionary natural selection process. Generations of solutions are evaluated according to a fitness value and only those candidates with high fitness values are used to create further solutions via crossover and mutation procedures. Both methods are valid and efficient methods in numeric programming and have been employed in various fields due to their strong convergence properties. Specific parameter settings for the algorithms are described in Table 2, where D is the dimension of the position. For the small scale problem, for example, $(J8, O27, M8)$, the maximum number of iterations is 200 in each trial. For other problems, the maximum number of iterations is 200. Each experiment (for each algorithm) was repeated 10 times with different random seeds. The average fitness values of the best solutions throughout the optimization run were recorded. The averages (f) and the standard deviations (std) were calculated from the 10 different trials. The standard deviation indicates the differences in the results during the 10 different trials. Usually another emphasis will be to generate the schedules at a minimal amount of time. So the completion time for 10 trials were used as one of the criteria to improve their performance.

Figs. 9, 10 and 11 illustrate the performance for the three algorithms during the search processes for the three FJSPs. Empirical results are illustrated in Table 3. In general. MPSO could be an ideal approach for solving the large scale problems when other algorithms failed to give a better solution.

Table 3. Comparing the results for FJSPs.

$Instance$	$Items$	GA	SPSO	MPSO
$(J8, O27, M8)$	Best	221	200	161
	average	246.6999	226.6999	180.3999
	std	$\pm$20.8856	$\pm$10.8263	$\pm$9.4148
$(J10, O30, M10)$	Best	139	133	74
	average	153.8000	142.1999	97.5000
	std	$\pm$8.1092	$\pm$6.8818	$\pm$12.7220
$(J15, O56, M10)$	Best	276	231	168
	average	292.2000	251.3999	196.1000
	std	$\pm$10.0677	$\pm$12.2572	$\pm$13.9316

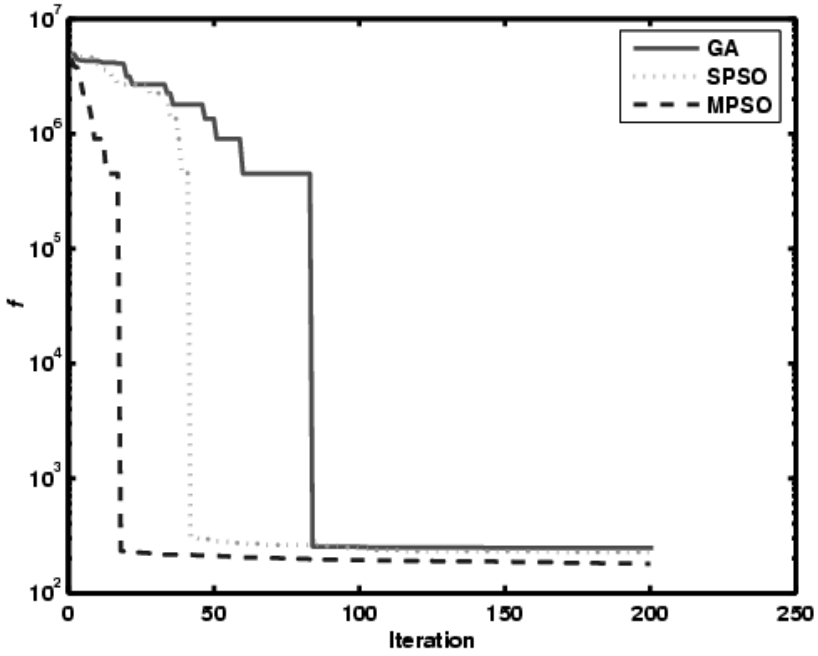

Figure 9. The performance of the algorithms for $(J8, O27, M8)$ FJSP.

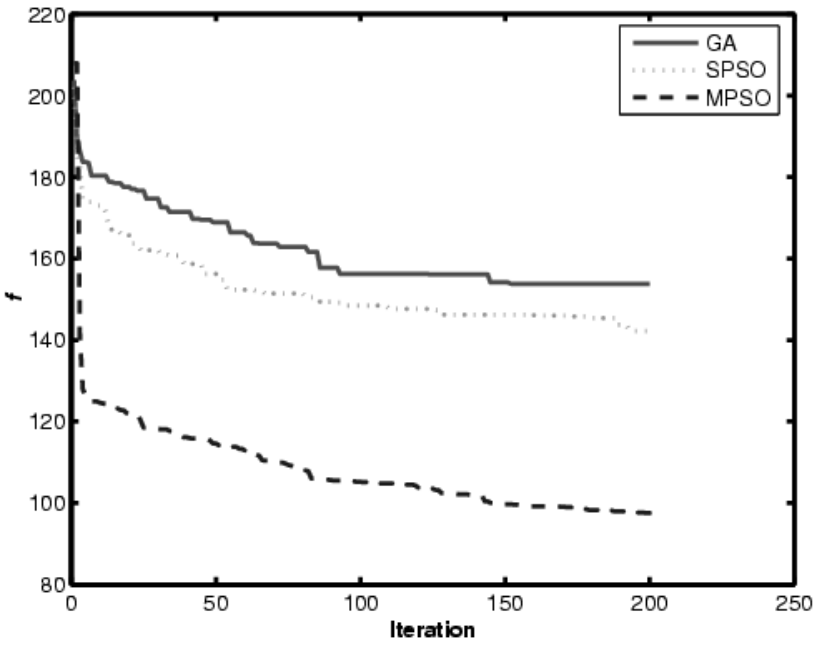

Figure 10. The performance of the algorithms for $(J10, O30, M10)$ FJSP.

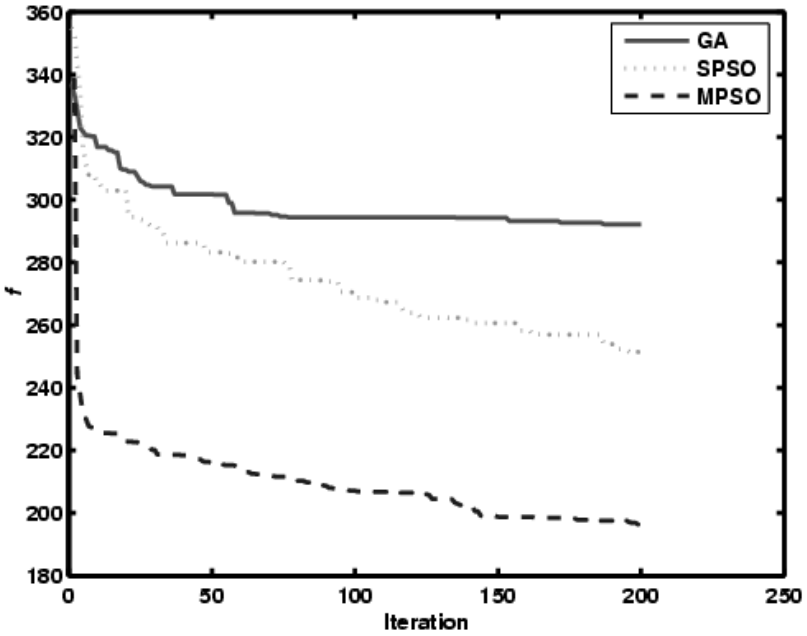

Figure 11. The performance of the algorithms for $(J15, O56, M10)$ FJSP.

7. Conclusions

Particle swarm optimization algorithm has exhibited good performance across a wide range of real world applications but not much work has been reported of its usage to solve the multi-objective Flexible Job-shop Scheduling Problems (FJSP) very well. Initial difficulty consists of how to map the problem solution to the particle space. In other words, encoding representation can be extremely important when trying to find solutions to the problem in the metaheuristic algorithm. The data structures, such as the particle position, plus the algorithm combine to make efficient programs. Better efficiency of search can be achieved by modifying the encoding representation and its related operators so as to generate feasible solutions and avoiding the use of a repair mechanism. A bad encoding representation can increase the size of the search space or slow down the algorithm if too many repair operators are needed to ensure the representation is valid. We have to be confronted with the second difficulty of how to ensures a good trade-off between exploration and exploitation in the algorithm.

In this paper, we modeled the scheduling problem for the multi-objective Flexible Job-shop Scheduling Problems (FJSP) and make an attempt to formulate and solve the problem using a multi-swarm approach. We extend the representations of the position and velocity of the particles in PSO. In our PSO algorithms, the position representation of the particles has two components: operation order and machine selection, and it is with a variable length strategy. In our encoding representations, we considered particle's position encoding as the binary representation of an integer. Bi-metrics Binary encoding approach is used efficiently. The different individual is separated into different group to map the operation order and machine selection respectively, which is favorable and reasonable. To match the two component

characteristics, we introduce a multi-swarm search algorithm. In the algorithm, all particles are clustered spontaneously into two different groups of the whole swarm. One is mapped for the operation order, and another for the machine selection. Each group consists of multiple sub-swarms. In the same group, every particle can connect to more than one sub-swarm, and a crossover neighborhood topology is constructed between different sub-swarms. The particles in the same sub-swarm would carry some similar functions as possible and search for their optimal. Each sub-swarm would approach its appropriate position (solution), which would be helpful for the whole swarm to keep in a good balance state. The proposed multi-swarm PSO algorithm is illustrated theoretically so that it converges with a probability of 1 towards the global optimum. The details of the implementation for the multi-objective FJSP are provided and its performance was compared using computational experiments. The empirical results have shown that the proposed algorithm is an available and effective approach for the multi-objective FJSP, especially for large scale problems.

References

[1] Kennedy, J., Eberhart, R.: *Swarm Intelligence*, Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[2] Clerc, M.: *Particle Swarm Optimization*, ISTE Publishing Company, London, 2006.

[3] Schute, J. F., Groenwold, A. A.: A study of global optimization using particle swarms, *Journal of Global Optimization*, **31**, 2005, 93–108.

[4] Grosan, C., Abraham, A., Nicoara, M.: Search optimization using hybrid particle sub-swarms and evolutionary algorithms, *International Journal of Simulation Systems, Science & Technology*, **6**(10–11), 2005, 60–79.

[5] Abraham, A., Guo, H., Liu, H.: Swarm intelligence: foundations, perspectives and applications, in: *Swarm Intelligent Systems*, (N. Nedjah and L. Mourelle, Eds.), *Studies in Computational Intelligence*, Springer Verlag Germany, 2006, 3–25.

[6] Abraham, A., Liu, H., Clerc, M.: Chaotic dynamic characteristics in swarm intelligence, *Applied Soft Computing Journal*, **7**, 2007, 1019–1026.

[7] Paquet, U., Engelbrecht, A. P.: Particle swarms for linearly constrained optimisation, *Fundamenta Informaticae*, **76**(1–2), 2007, 147–170.

[8] Das, S., Abraham, A., Konar, A.: Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm, *Pattern Recognition Letters*, **29**, 2008, 688–699.

[9] Liu, H., Abraham, A., Hassanien, A.E.: Scheduling job on computational Grids using fuzzy particle swarm algorithm, *Future Generation Computer Systems*, `http://dx.doi.org/10.1016/j.future.2009.05.022`, 2009.

[10] Garey, M. R., Johnson, D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[11] Brucker, P.: *Scheduling Algorithms* (2nd edition), Springer, Heidelberg, 1998.

[12] Pinedo, M. L.: *Scheduling: Theory, Algorithms, and Systems* (3rd edition), Springer, Heidelberg, 2008.

[13] Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines, *Computing*, **45**(4), 1990, 369–375.

[14] Jansen, K., Mastrolilli, M., Solis-Oba, R.: Approximation algorithms for flexible job shop problems. *Lecture Notes in Computer Science*, **1776**, 2000, 68–77.

[15] Mastrolilli, M., Gambardella, L. M.: Effective neighborhood functions for the flexible job shop problem, *Journal of Scheduling*, **3**(1), 2002, 3–20.

[16] Pezzellaa, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem, *Computers and Operations Research*, **35**(10), 2008, 3202–3212.

[17] Stecke, K. S.: Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems, *Management Science*, **29**(3), 1983, 273–288.

[18] Akella, R., Choong, Y.: Performance of a hierarchical production scheduling policy, *IEEE Transactions on Components, Hybrids and Manufacturing Technology*, **7**(3), 1984, 225–248.

[19] Bona, B., Brandimarte, P.: Hybrid hierarchical scheduling and control systems in manufacturing, *IEEE Transactions on Robotics and Automation*, **6**(6), 1990, 673–686.

[20] Brandimarte, P.: Routing and scheduling in a flexible job-shop by tabu search, *Annals of Operations Research*, **2**, 1993, 158–183.

[21] Paulli, J.: A hierarchical approach for the FMS scheduling problem, *European Journal of Operational Research*, **86**(1), 1995, 32–42.

[22] Barnes, J. W., Chambers, J. B.: Flexible job shop scheduling by tabu search, Graduate program in operations research and industrial engineering, *Technical Report*, ORP 9609, University of Texas, Austin, 1996, `http://www.cs.utexas.edu/users/jbc/`

[23] Hurink, J., Jurish, B., Thole, M.: Tabu search for the job shop scheduling problem with multi-purpose machines, *OR-Spektrum*, **15**, 1994, 205–215.

[24] Dauzére-Pérés S., Paulli, J.: An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, *Annals of Operations Research*, **70**, 1997, 281–306.

[25] Mastrolilli, M., Gambardella, L. M.: Effective neighborhood functions for the flexible job shop problem, *Journal of Scheduling*, **3**, 1996, 3–20.

[26] Saidi-Mehrabad, M., Fattahi, P.: Flexible job shop scheduling with tabu search algorithms, *International Journal of Advanced Manufacturing Technology*, **32**, 2007, 563–570.

[27] Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation, *International Journal of Computers and Industrial Engineering*, **30**, 1996, 983–997.

[28] Cheng, R., Gen M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies, *International Journal of Computers and Industrial Engineering*, **36**, 1996, 343–364.

[29] Gao, J., Sun, L., Gen, M.: A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems, *Computers and Operations Research*, **35**(9), 2008, 2892–2907.

[30] Zhang, H., Gen, M.: Multistage-based genetic algorithm for flexible job-shop scheduling problem, *Journal of Complexity International*, **11**, 2005, 223–232.

[31] Chen, H., Ihlow, J., Lehmann, C.: Agenetic algorithm for flexible job-shop scheduling, *IEEE International Conference on Robotics and Automation*, Detroit, 1999, 1120–1125.

[32] Jia, H. Z., Nee, A., Fuh, J., Zhang, Y.: A modified genetic algorithm for distributed scheduling problems, *International Journal of Intelligent Manufacturing*, **14**, 2003, 351–362.

[33] Ho, N. B., Tay, J. C.: GENACE: an efficient cultural algorithm for solving the flexible job-Shop problem, *IEEE International Conference on Robotics and Automation*, 2004, 1759–1766.

[34] Ho, N. B., Tay, J. C., Lai, E.: An effective architecture for learning and evolving flexible job-shop schedules, *European Journal of Operational Research*, **179**, 2007, 316–333.

[35] Ong, Z. X., Tay, J. C., Kwoh, C. K.: Applying the clonal selection principle to find flexible job-shop schedules. *Lecture Notes in Computer Science*, **3627**, 2005, 442–455.

[36] Kleeman, M. P., Lamont, G. B.: Scheduling of flow-shop, job-shop, and combined scheduling problems using MOEAs with fixed and variable length chromosomes, *Studies in Computational Intelligence*, **49**, 2007, 49–99.

[37] Kacem, I., Hammadi, S., Borne, P.: Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Transactions on Systems, Man, and Cybernetics*, Part C, **32**(1), 2002, 1–13.

[38] Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic, *Mathematics and Computers in Simulation*, **60**, 2002, 245–276.

[39] Tanev, I. T., Uozumi, T., Morotome, Y.: Hybrid evolutionary algorithm-based real-world flexible job shop scheduling problem: application service provider approach, *Applied Soft Computing*, **5**, 2004, 87–100.

[40] Liouane, N., Saad, I., Hammadi, S., Borne, P.: Ant systems & local search optimization for flexible job shop scheduling production, *International Journal of Computers, Communications & Control*, **2**(2), 2007, 174–184.

[41] Blum C., Samples, M.: An Ant Colony Optimization Algorithm for Shop Scheduling Problems, *Journal of Mathematical Modeling and Algorithms*, **3**, 2004, 285–308.

[42] Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job-shop problem, *Management Science*, **42**(2), 1996, 797–813.

[43] Wu, Z., Weng, M. X.: Multiagent scheduling method with earliness and tardiness objectives in flexible job shops, *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, **35**(2), 2005, 293–301.

[44] Xia, W., Z. Wu, Z.: An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems, *Computers and Industrial Engineering*, **48**, 2005, 409–425.

[45] Sha, D. Y., Hsu, C. Y.: A hybrid particle swarm optimization for job shop scheduling problem, *Computers & Industrial Engineering*, **51**(4), 2006, 791–808.

[46] Giffler, J. and Thompson, G. L.: Algorithms for solving production scheduling problems, *Operations Research*, **8**, 1960, 487–503.

[47] Baykasoğlu, A., Özbkir, L., Sönmez, A.: Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems, *Journal of Intelligent Manufacturing*, **15**, 2004, 777–785.

[48] Parsopoulos, K. E., Vrahatis, M. N.: Recent approaches to global optimization problems through particle swarm optimization, *Natural Computing*, **1**, 2002, 235–306.

[49] Ripon, K., C. Tsang, C., Kwong, S. An evolutionary approach for solving the multi-objective job-shop scheduling problem, *Studies in Computational Intelligence*, **49**, 2007, 165–195.

[50] Salman, A., Ahmad, I., Al-Madani, S.: Particle swarm optimization for task assignment problem, *Microprocessors and Microsystems*, **26**, 2002, 363–371.

[51] van den Bergh, F., Engelbrecht, A. P.: A study of particle swarm optimization particle trajectories, *Information Sciences*, **176**, 2006, 937–971.

[52] Liu, H., Abraham, A.: Chaos and swarm intelligence, in: *Intelligent Computing Based on Chaos* (L. Kocarev, Z. Galias, S. Lian, Eds.), *Studies in Computational Intelligence*, Springer Verlag, Germany, **184**, 2009, 197–212.

[53] Clerc M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multidimensional complex space, *IEEE Transactions on Evolutionary Computation*, **6**, 2002, 58–73.

[54] Cristian, T. I.: The particle swarm optimization algorithm: convergence analysis and parameter selection, *Information Processing Letters*, **85**(6), 2003, 317–325.

[55] Liu, H., Li, B., Ji, Y., Sun, T.: Particle swarm optimisation from lbest to gbest, in: *Applied Soft Computing Technologies: The Challenge of Complexity* (A. Abraham, B. D. Baets, M. Köppen, B. Nickolay Eds.), Springer, 2006, 537–545.

[56] Liu, H., Abraham, A.: An hybrid fuzzy variable neighborhood particle swarm optimization algorithm for solving quadratic assignment problems, *Journal of Universal Computer Science*, **13**(7), 2007, 1032–1054.

[57] Gen, M., Tsujimura, Y., Kubota, E.: Solving job-shop scheduling problem using genetic algorithms, in: *Proceedings of the 16th International Conference on Computer and Industrial Engineering*, 1994, 576–579.

[58] Liu, B., Wang, L., Jin, Y.: An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers, *Computers and Operations Research*, **33**(10), 2006, 2960–2971.

[59] Tasgetiren, M. F., Sevkli, M., Liang, Y. C., Gencyilmaz, G.: Particle swarm optimization algorithm for permutation flowshop sequencing problem, *Lecture Notes in Computer Science*, **3172**, 2004, 382–389.

[60] van den Bergh, F., Engelbrecht, A. P.: A cooperative approach to particle swarm optimization, *IEEE Transactions on Evolutionary Computation*, **8**(3), 2004, 225–239.

[61] Settles, M., Rylander, B.: Neural network learning using particle swarm optimizers, *Advances in Information Science and Soft Computing*, 2002, 224–226.

[62] Blackwell, T., Branke, J.: Multi-swarm optimization in dynamic environments, *Lecture Notes in Computer Science*, **3005**, 2004, 489–500.

[63] Settles, M., Soule, T.: Breeding swarms: a GA/PSO hybrid, in: *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO), 2005, 161–168.

[64] Elshamy, W., Emara, H. M., Bahgat, A.: Clubs-based particle swarm optimization, in: *Proceedings of the IEEE International Conference on Swarm Intelligence Symposium*, **1**, 2007, 289–296.

[65] Guo, C., Tang, H.: Global convergence properties of evolution stragtegies, *Mathematica Numerica Sinica*, **23**(1), 2001, 105–110.

[66] He, R., Wang, Y., Wang, Q., Zhou, J., Hu, C.: An improved particle swarm optimization based on self-adaptive escape velocity, *Chinese Journal of Software*, **16**(12), 2005, 2036–2044.

[67] Weisstein, E. W.: Borel-Cantelli Lemma. From MathWorld – A Wolfram Web Resource, `http://mathworld.wolfram.com/Borel-CantelliLemma.html`, 2007.

[68] Xu, Z., Cheng, G., Liang, Y.: Search capability for an algebraic crossover, *Journal of Xi'an Jiaotong University*, **33**(10), 1999, 88–99.

[69] Whitley, L. D.: Fundamental principles of deception in genetic search, in: *Foundation of Genetic Algorithms*, (J. E. Gregory, Ed.), California: Morgan Kaufmann Publishers, 1991, 221–241.

[70] Holland, J. H.: *Adaptation in natural and artificial systems*, Ann Arbor: University of Michigan Press, 1975.

[71] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[72] Taillard, E. D.: Benchmarks for basic scheduling problems, *European Journal of Operational Research*, **64**(2), 1993, 278–285.

[73] Tay, J. C., Wibowo, D.: An effective chromosome representation for evolving flexible job shop schedules, *Lecture Notes in Computer Science*, **3103**, 2004, 210–221.

[74] Fisher, H., Thompson, L.: Probabilistic learning combinations of local job shop scheduling rules, in: *Industrial Scheduling* (J. F. Muth, G. L. Thompson, Eds.), NJ: Prentice-Hall, 1968, 225–251.

[75] Lawrence, S.: *Supplement To Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, 1984.

[76] Abraham, A., Nedjah, N., Mourelle, L.: Evolutionary computation: from genetic algorithms to genetic programming, in: *Studies in Computational Intelligence* (N. Nedjah, et al. Eds.), Germany: Springer Verlag, 2006, 1–20.

[77] Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic publishers, Netherland, 2000.